Power-aware Deep Learning Model Serving with μ-Serve

论文信息

Title: Power-aware Deep Learning Model Serving with μ-Serve

Authors:Haoran Qiu, Weichao Mao, Archit Patke, and Shengkun Cui,

University of Illinois Urbana-Champaign;

Saurabh Jha, Chen Wang, and Hubertus Franke, IBM Research；

Zbigniew Kalbarczyk, Tamer Başar;

and Ravishankar K. Iyer, University of Illinois Urbana-Champaign

Conference: ATC'24', July 10–12, 2024 • Santa Clara, CA, USA

研究背景

1.随着大型深度学习模型服务工作负载的日益普及，迫切需要降低模型服务集群的能耗，同时保持满意的吞吐量或模型服务延迟要求。

2.模型复用方法，如模型并行、模型放置、复制和批处理，旨在优化模型服务性能。

3.然而，这些方法未能利用 GPU 频率缩放来节省能源。

挑战

1.首先，确定满足性能要求的同时最大化能源效率的 GPU 频率是一个挑战。

2.其次，服务非确定性生成模型使得难以区分 GPU 频率下调的负面影响与执行时间固有的非确定性之间的区别。

3.最后，与 CPU 不同，GPU 不支持细粒度（例如，每个核心）频率缩放。

模型服务系统

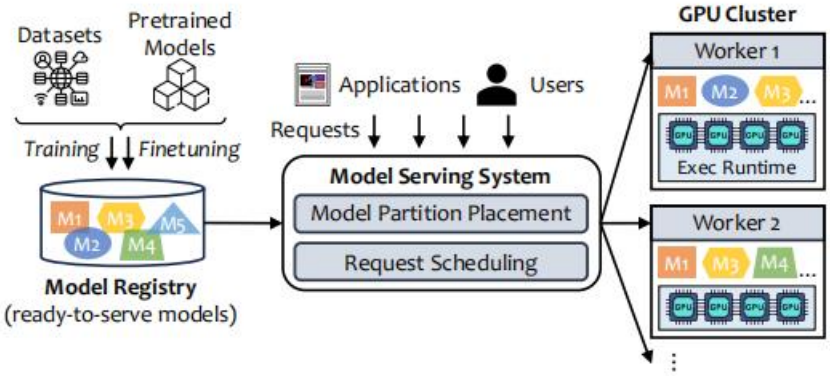


Figure 1: Model training and model serving.

节能机会

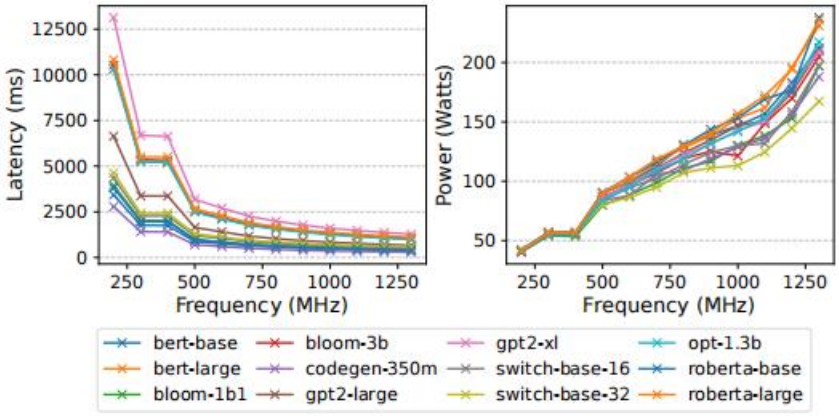


Figure 2: Model-serving latency and power consumption characterization at varying GPU core frequency levels.

自回归模式

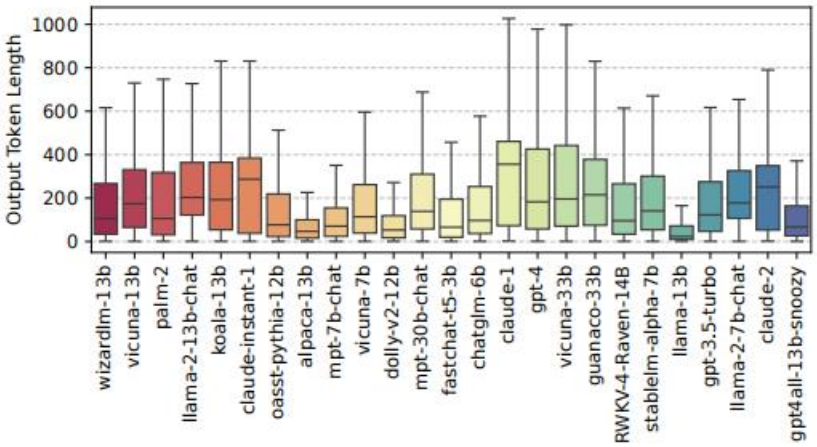


Figure 3: Output token length distributions of various large language models collected in the LMSYS-Chat-1M dataset.

$$T = C + K * N$$

T：请求的执行时间。C：模型服务系统的开销，包括 DNS 查找、代理、排队和输入分词。K：生成一个标记的延迟。

μ-Serve 设计

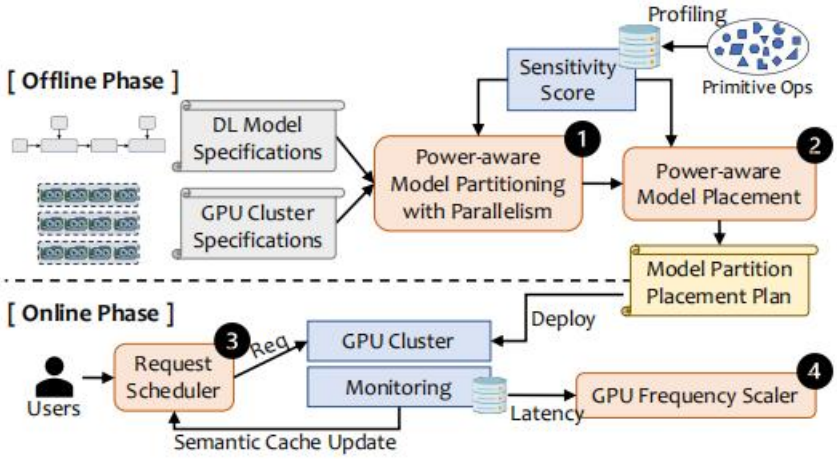


Figure 4: μ-Serve architecture overview and workflow.

节能型模型并行

关键见解 1：一些算子对 GPU 频率变化更敏感（即敏感算子），而其他算子则不敏感（即不敏感算子）。

关键见解 2：不敏感算子通常对服务请求的端到端延迟贡献较小。由于模型分区

可以容忍的最低 GPU 频率由最敏感的算子决定，因此 μ-Serve 避免将敏感和不敏感算子混合到同一个分区中。

算子敏感性评分分析

1. 我们将每个算子的敏感评分定义为Δ延迟/Δ频率。

2. 我们独立运行每个算子，每个张量维度重复 5000 次，并取所有运行的平均值。

3. 我们采用阿里巴巴发布的 AI-Matrix 生产数据集中的张量维度。

调度

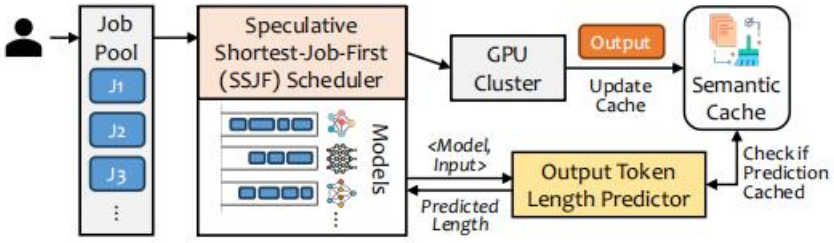


Figure 5: Workflow of the speculative scheduling in μ-Serve.

输出标记长度预测器

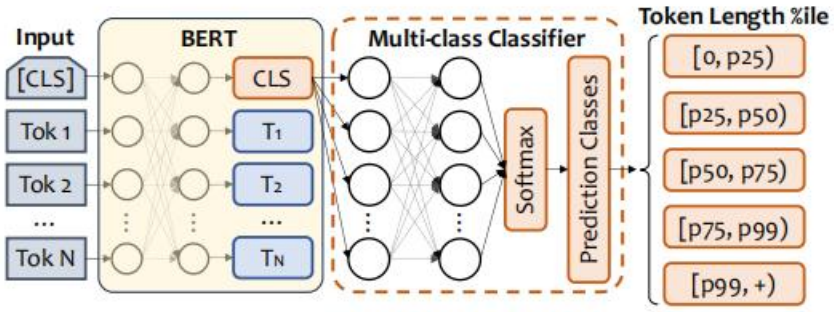


Figure 6: Output token length predictor in μ-Serve.

GPU 频率缩放

乘法增加加法减少（MIAD）算法

Algorithm 3 GPU Frequency Scaling

Require: GPU Device GD, Model Partitions MP (a map of ⟨model, partition⟩ pairs placed on GD)

```
1:  procedure GPUFREQUENCYSCALING(GD, MP)
2:      // Start from the default high frequency
3:      freq = DEFAULT
4:      GD.setFrequency(freq)
5:      while True do
6:          action = SAME    ▷ one of {UP, DOWN, SAME}
7:          for each p in MP do
8:              target = p.getModelSLO()
9:              actual = p.getModelLatency()
10:             slack = (target − actual)/target
11:             if slack < 0.05 then
12:                 action = UP          ▷ requires up-scaling
13:                 break
14:             end if
15:             latency = p.getWeight() * actual
16:             degradation = latency ∗ STEP/freq
17:             if degradation/target < slack − 0.05 then
18:                 action = DOWN            ▷ down-scaling
19:             end if
20:         end for
21:         if action==UP then
22:             freq = min(MAX, freq ∗ 2)
23:         else if action==DOWN then
24:             freq = max(MIN, freq − STEP)
25:         end if
26:         GD.setFrequency(freq)
27:     end while
28: end procedure
```

实验设置

工作负载：我们使用 Microsoft Azure 函数跟踪和 SenseTime 私有数据中心专门用于 ML 工作负载的生产跟踪来驱动推理工作负载。

指标：我们使用服务水平目标达成率（即，在延迟服务水平目标内服务的请求百分比）和功耗（以瓦特为单位）作为主要评估指标。

测试平台：我们在具有 8 个节点和 16 个 GPU 的集群上部署 μ-Serve。每个节点是 IBM Cloud gx2-16x128x2v100 实例，配备 2 个 NVIDIA Tesla V100（16GB）GPU。每个 GPU 支持的最大流多处理器（SM）频率为 1380 MHz，最小为 200 MHz。

Table 1: Details of the models used in experiments. *The latency is measured for a single query with a sequence length of 512 on a single GPU. AR stands for autoregressive.*

Model	# of Params	Size	Latency	AR?
ResNet-50	25M	0.2 GB	51 ms	No
BERT-base	110 M	0.5 GB	123 ms	No
BERT-large	340 M	1.4 GB	365 ms	No
RoBERTa-base	125 M	0.5 GB	135 ms	No
RoBERTa-large	355 M	1.4 GB	382 ms	No
OPT-1.3b	1.3 B	5.0 GB	1243 ms	Yes
OPT-2.7b	2.7 B	10.4 GB	2351 ms	Yes
GPT2-large	774 M	3.3 GB	832 ms	Yes
GPT2-xl	1.5 B	6.4 GB	1602 ms	Yes
CodeGen-350m	350 M	1.3 GB	357 ms	Yes
CodeGen-2b	2.0 B	8.0 GB	2507 ms	Yes
Bloom-1b1	1.1 B	4.0 GB	523 ms	Yes
Bloom-3b	3.0 B	11.0 GB	1293 ms	Yes
Switch-base-16	920 M	2.4 GB	348 ms	Yes
Switch-base-32	1.8 B	4.8 GB	402 ms	Yes

端到端结果

节能

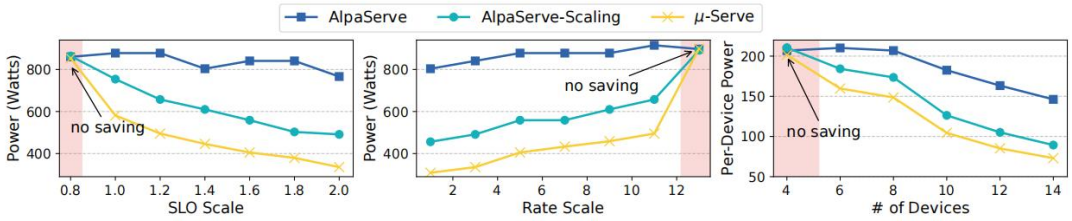


Figure 7: Power saving evaluation.

模型服务性能

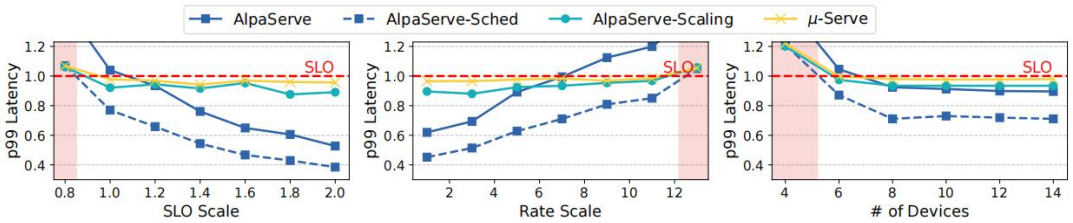


Figure 8: SLO preservation evaluation.

调度

标记长度预测器评估

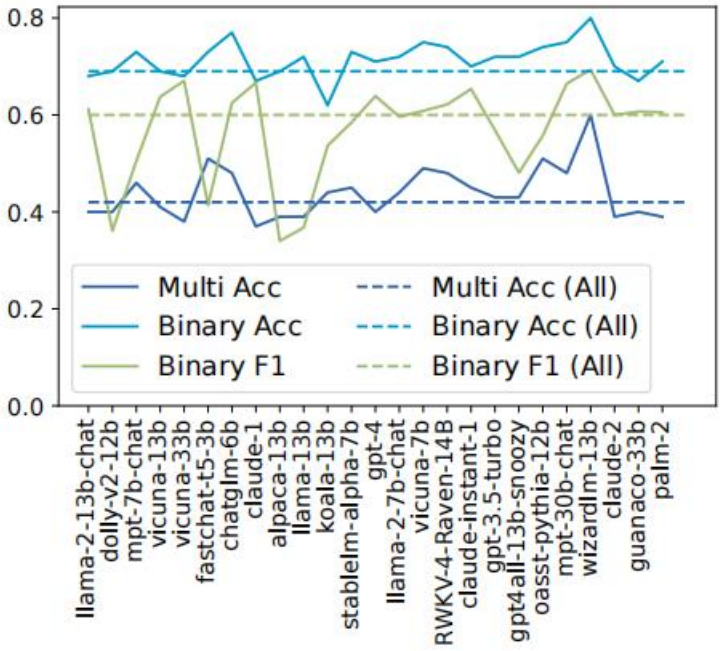


Figure 11: Predictor accuracy.

调度器评估

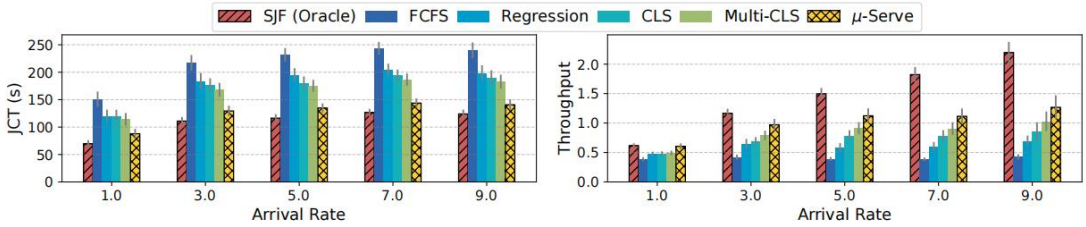


Figure 9: Scheduler evaluation across varying request arrival rates.

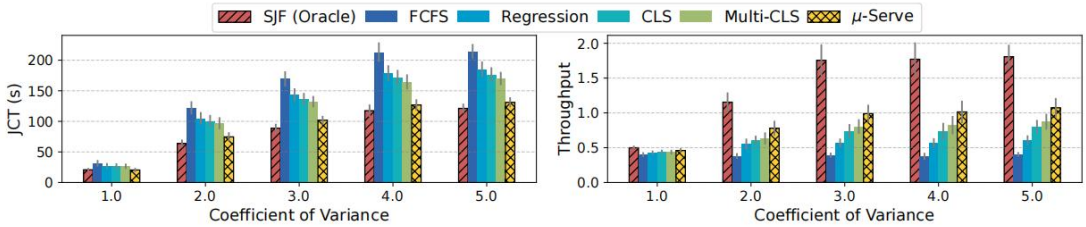


Figure 10: Scheduler evaluation across varying request burstiness.

预测器开销

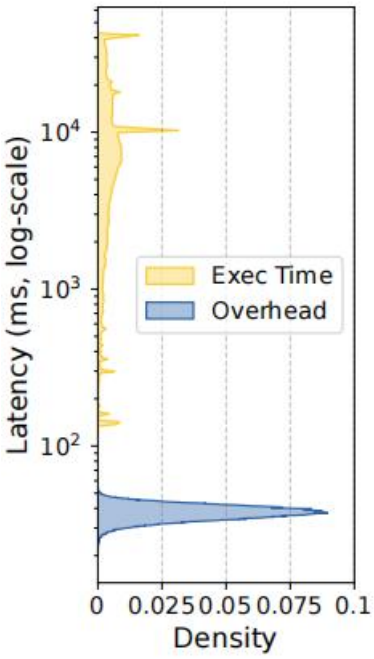
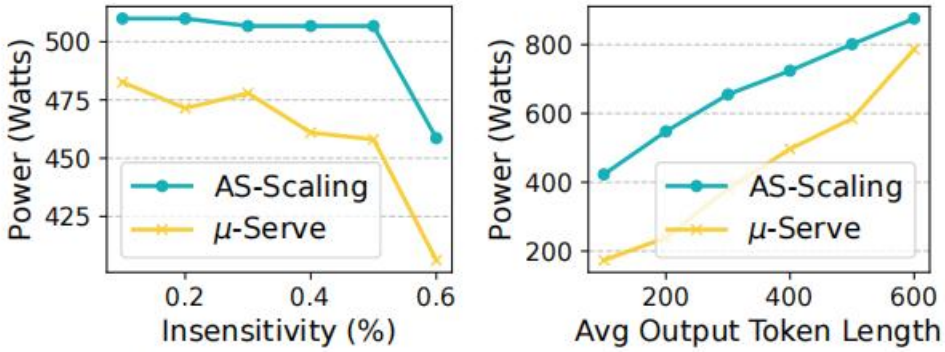


Figure 12: Overhead.

模型分区和放置



(a) Insensitivity (%). **(b)** Output variation.

Figure 14: Robustness analysis.

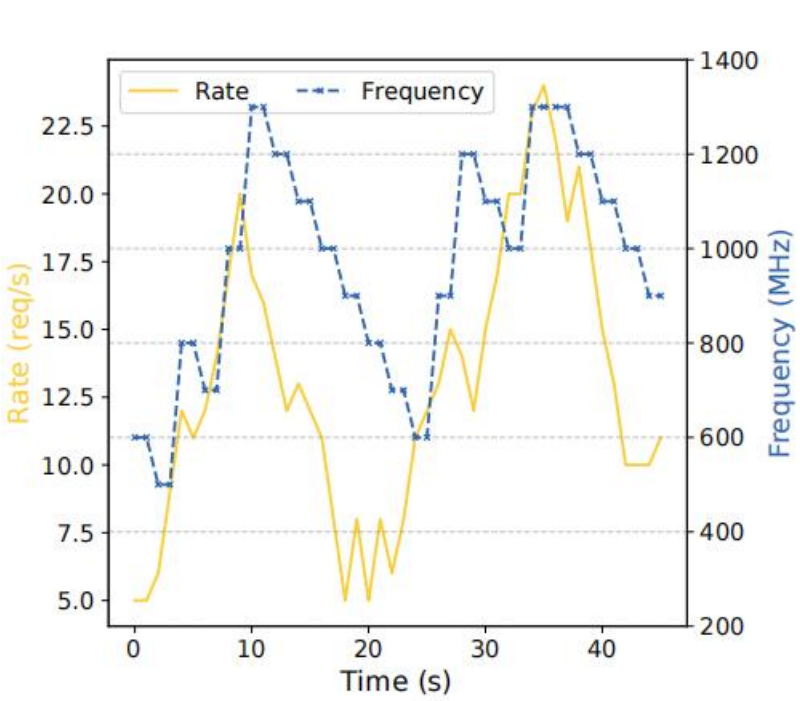


Figure 13: Frequency scaling.

讨论和未来挑战

异构加速器：集群中的 GPU 设备在硬件（例如，A100、V100 和 T4）、资源配置（例如，内存大小）、频率范围方面可能是异构的。

其他电源管理功能：更先进的 GPU 上有其他电源管理功能，如功率上限、各种 GPU 操作模式、内存频率缩放和 MIG 共享。

可扩展性：图 7 显示，随着集群在没有空闲设备的情况下进行扩展，μ-Serve 始终促进每个设备的节能。基于这些结果，我们断言，在确保服务水平目标达成的同时，云规模部署中也可能存在类似的节能机会。

结论

1. μ-Serve 是一个用于服务多个大型深度学习模型的系统，它最大限度地节省能源，同时保持请求服务的服务水平目标（SLO）达成。

2. μ-Serve 的关键创新在于其对算子敏感性的细粒度建模和节能型模型配置，这为节能创造了机会。

3. 这些节能机会可以通过在线动态 GPU 频率缩放来利用，从而在不违反不同条件下的服务水平目标的情况下实现节能，正如通过广泛评估所量化和讨论的那样。